

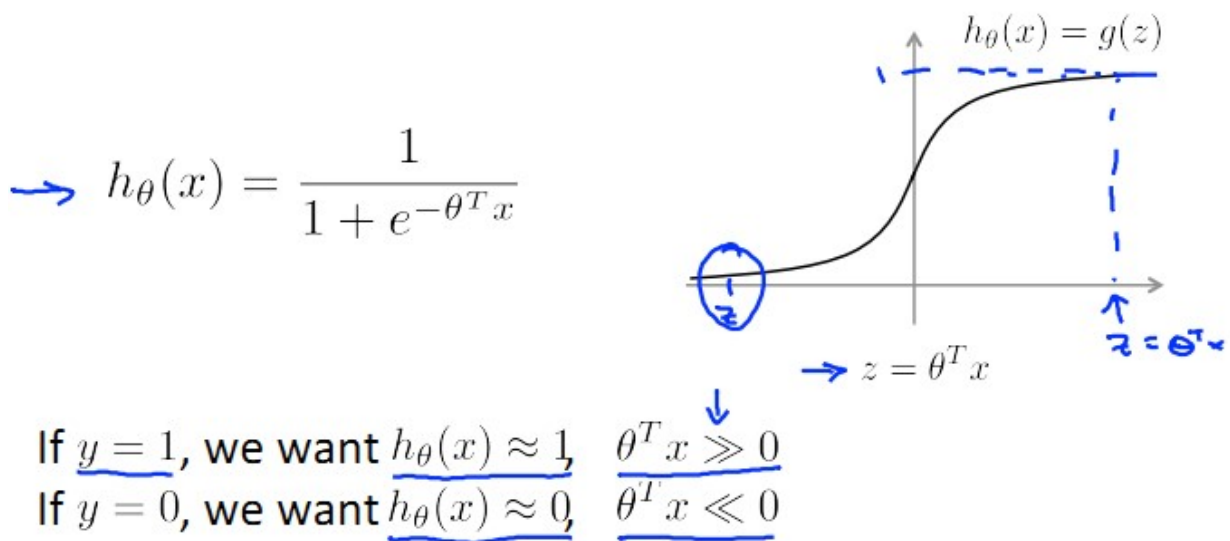
ML 07: SUPPORT VECTOR MACHINE

이번시간에 *Support Vector Machine, SVM* 을 배운다.

Optimization Objective

먼저 직관을 얻기 위해 *logistic regression* 의 *sigmoid function* 을 좀 보자.

Alternative view of logistic regression



(<http://blog.csdn.net/abcjennifer>)

$y = 1$ 이면 $\theta^T x \gg 0$ 이어야 $h(x)$ 가 1 에 가까워 진다.

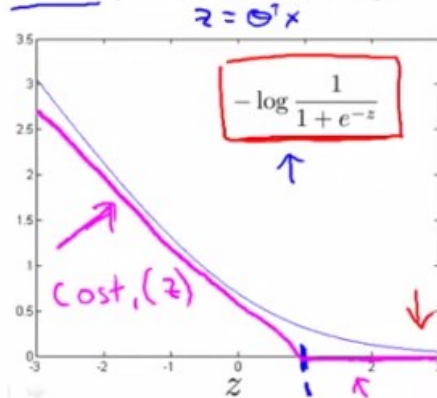
이제 *cost function* 에 $h(x)$ 를 넣자. 그리고 $m = 1$ 인 트레이닝 셋에 대해서 보면

Alternative view of logistic regression (x, y)

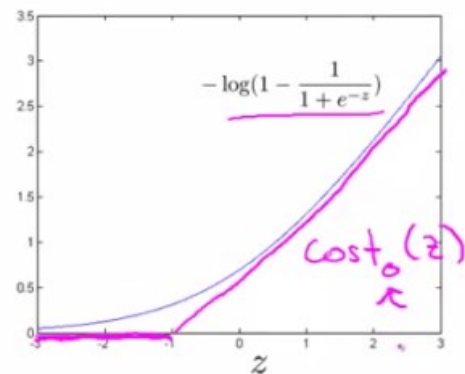
Cost of example: $-(y \log h_{\theta}(x) + (1 - y) \log(1 - h_{\theta}(x)))$ ←

$$= -y \log \frac{1}{1 + e^{-\theta^T x}} - (1 - y) \log \left(1 - \frac{1}{1 + e^{-\theta^T x}}\right)$$

If $y = 1$ (want $\theta^T x \gg 0$):



If $y = 0$ (want $\theta^T x \ll 0$):



(<http://blog.csdn.net/abcjennifer>)

파란 그래프에서 볼 수 있듯이 $y = 1$ 일때 $\theta^T x \gg 0$ 이면 $cost$ 가 상당히 낮아지는걸 볼 수 있다. 이 그래프를 좀 단순화 해서 자주색 그래프를 만들어 보자. 두개의 직선으로 만들었는데, 이 $cost$ function 을 계산하면 상당히 근접한 값을 얻을 수 있고, 동시에 그래프가 단순해져 $computational$ advantage 를 얻을 수 있다.

각각 좌측, 우측에 있는 $cost$ function 을 이렇게 쓴다.

$$cost_1(z) \quad (y = 1)$$

$$cost_0(z) \quad (y = 0)$$

logistic regression 식 에서 $-\log h(x)$ 를 $cost_1(z)$ 로, $-\log(1 - h(x))$ 를 $cost_0(z)$ 로 바꾸면

$$\min_{\theta} \frac{1}{m} \left[\sum_{i=1}^m y^{(i)} (-\log h_{\theta}(x^{(i)})) + (1 - y^{(i)}) (-\log(1 - h_{\theta}(x^{(i)}))) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

$$\begin{aligned} cost_1(z) \quad (y = 1) \\ cost_0(z) \quad (y = 0) \end{aligned}$$

$$\begin{aligned} cost_1(z) \quad (y = 1) \\ cost_0(z) \quad (y = 0) \\ \min_{\theta} \frac{1}{m} \left[\sum_{i=1}^m y^{(i)} cost_1(\theta^T x) + (1 - y^{(i)}) (cost_0(\theta^T x)) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2 \end{aligned}$$

이 때, $1/m$ 은 상수이므로 제거해도 어차피 똑같은 $\theta(\text{theta})$ 를 얻을 수 있다.

그리고 식을 좀 간략히 적어보면

$$\min_{\theta} A + \lambda B$$

여기서 λ 가 하는 일은 *low cost* ('A')와 *small parameter* ('B')를 조절하는 일이다. 식을 좀 변경하면 이렇게도 볼 수 있다. 여기서 C 는 $1 / \lambda$ 과 같은 역할이라 보면 된다.

$$\min_{\theta} C + \lambda B$$

아주 작은 수의 λ 를 사용하면 파라미터 B 가 커지는데, 이것은 C 가 커져 A 를 낮추고 B 를 높이는 것과 똑같다. 반대로 C 가 작으면 A 가 커지고, B 가 작아진다.

결국 C 를 쓰느냐 λ 를 쓰느냐는, 어떤 항을 옵티마이제이션의 중심으로 두느냐다. 최적화된 파라미터를 찾는건 똑같다.

식을 마지막으로 정리하면,

$$\min_{\theta} C \left[\sum_{i=1}^m y^{(i)} cost_1(\theta^T x) + (1 - y^{(i)}) (cost_0(\theta^T x)) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

결국 위 식 (*cost*)를 최소화 하면, $y = 1$ 일때 $\theta^T x \gg 0$ 이 되므로 $h(x) == 1$ 이란 뜻이다.

SVM hypothesis

$$\rightarrow \min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

Hypothesis:

$$h_{\theta}(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

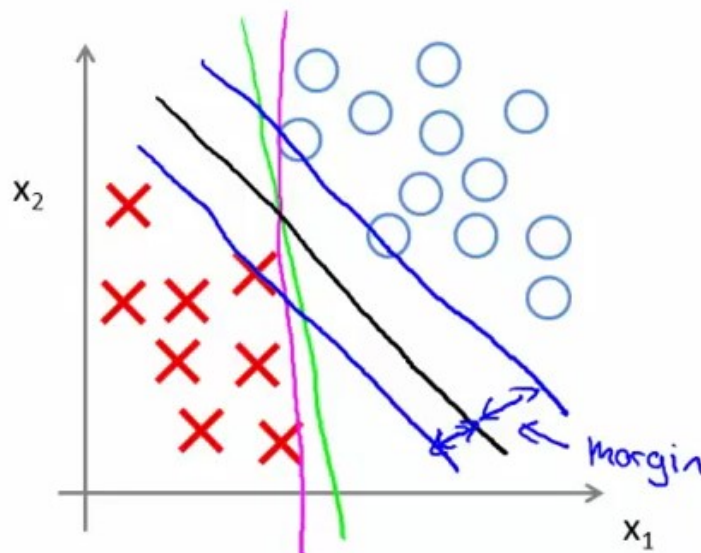
(<http://blog.csdn.net/abcjennifer>)

Large Margin Intuition

SVM은 *large margin classifier*라 부르기도 한다. 왜 그런게 한번 살펴보자.

두 집단을 구분하는 초록색, 자주색, 검은색 직선을 생각해 보자.

SVM Decision Boundary: Linearly separable case



Large margin classifier

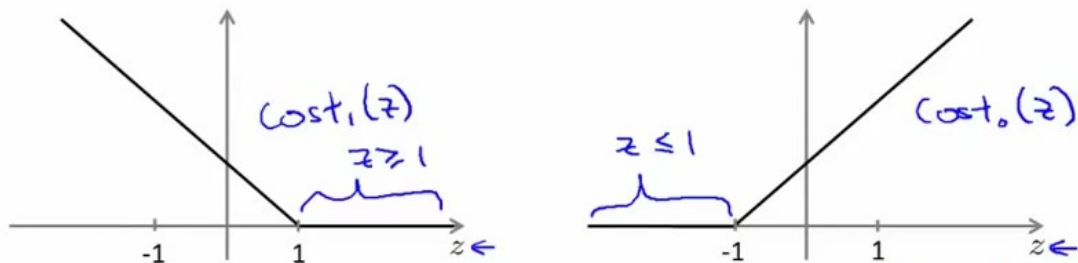
(<http://blog.csdn.net/abcjennifer>)

검은색 선이 가장 낮고, 자주색과 초록색은 두 집단을 분리하긴 하는데 썩 만족할만하
게는 아니다. 검은 선과 평행하고 각 점까지의 거리가 최소인 파란선을 그리자. 이걸
margin 이라 부른다. 다시 말해서 *margin* 이 클수록 좋은 *classification* 이다.

large margin 하고 *SVM* 하고 무슨 상관일까? 그 전에 먼저 **C** 를 좀 살펴보자.

Support Vector Machine

$$\rightarrow \min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



→ If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

→ If $y = 0$, we want $\theta^T x \leq -1$ (not just < 0)

$$\theta^T x \geq 1$$

$$\theta^T x \leq -1$$

$$C = 100,000$$

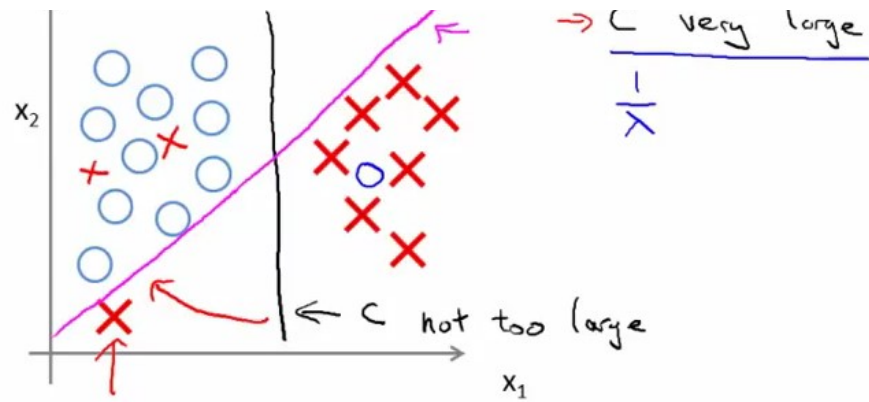
(<http://blog.csdn.net/abcjennifer>)

$z == \theta^T x$, 의 범위를 생각해 보면 $y = 1$ 일때 $z \geq 1$ 이길 바란다. 반대로
 $y = 0$ 이면 $z \leq -1$ 이면 $h(x)$ 로 충분히 만족할 만한 값을 얻을 수 있다.

이 때 **C** 가 매우 크면 **A** 즉, 아래의 식은 굉장히 작아진다. 거의 0 에 가깝게

$$\sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^T x) + (1 - y^{(i)}) (\text{cost}_0(\theta^T x))$$

Large margin classifier in presence of outliers



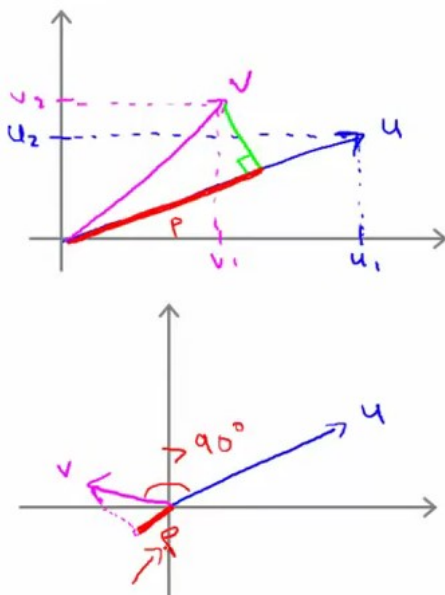
(<http://blog.csdn.net/abcjennifer>)

두 집단에 대해서 C 가 매우 크면, 다시 말해 A 가 0 에 가까우면 *overfitting* 된다 볼 수 있으므로 자주색과 비슷한 라인을 찾아낸다. 자주색 선은 모든 샘플에 대해 *large margin*을 가지고 있지만 그렇게 썩 좋은 *classification*이라 볼 수는 없다.

그러나 C 가 그렇게 크지 않으면 비 정상적인 샘플들은 조금 무시하고 검은색 선을 찾아낸다. 이게 *SVM*이 작동하는 방식이다.

Mathematics Behind Large Margin Classification

Vector Inner Product



$$\Rightarrow u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \Rightarrow v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$u^T v = ? \quad [u_1 \ u_2] \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$\|u\| = \text{length of vector } u \\ = \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$$

$$p = \text{length of projection of } v \text{ onto } u. \\ \text{Signed } u^T v = p \cdot \|u\| \leftarrow = v^T u \\ = u_1 v_1 + u_2 v_2 \leftarrow p \in \mathbb{R}$$

$$u^T v = p \cdot \|u\|$$

$$p < 0$$

Andrew B

(<http://blog.csdn.net/abcjennifer>)

SVM Decision Boundary

$$\omega = (\sqrt{\omega})^2$$

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 = \frac{1}{2} (\theta_1^2 + \theta_2^2) = \frac{1}{2} (\sqrt{\theta_1^2 + \theta_2^2})^2 = \frac{1}{2} \|\theta\|^2$$

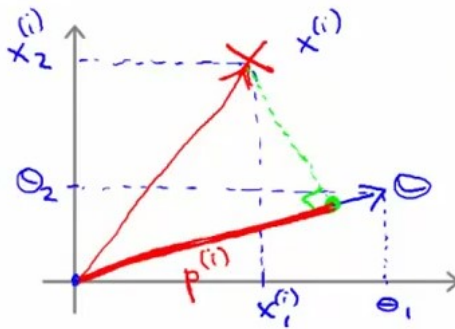
s.t. $\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$
 $\rightarrow \theta^T x^{(i)} \leq -1$ if $y^{(i)} = 0$
 Simplification: $\theta_0 = 0$. $n=2$

$$= \|\theta\|$$

$$\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \theta_0 = 0$$

$$\theta^T x^{(i)} = ?$$

$\uparrow \quad \uparrow$
 $u^T v$



$$\theta^T x^{(i)} = p^{(i)} \cdot \|\theta\|$$

$$= \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}$$

Andrew T

(<http://blog.csdn.net/abcjennifer>)

결국 C 가 아주 클 때 $A = 0$ 이므로 SVM cost function 을 최소화 하는 것은 아래 식 과 동일하다. 그런데 이 식을 풀어 보면

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

$$= \frac{1}{2} \|\theta\|^2$$

그리고 θ (theta) 와 x 를 벡터이므로 $\theta^T x^{(i)} = p^{(i)} * \|\theta\|$ 라 볼 수 있다. (여기서 $p^{(i)}$ 는 x 의 θ 로의 projection 된 선의 길이)

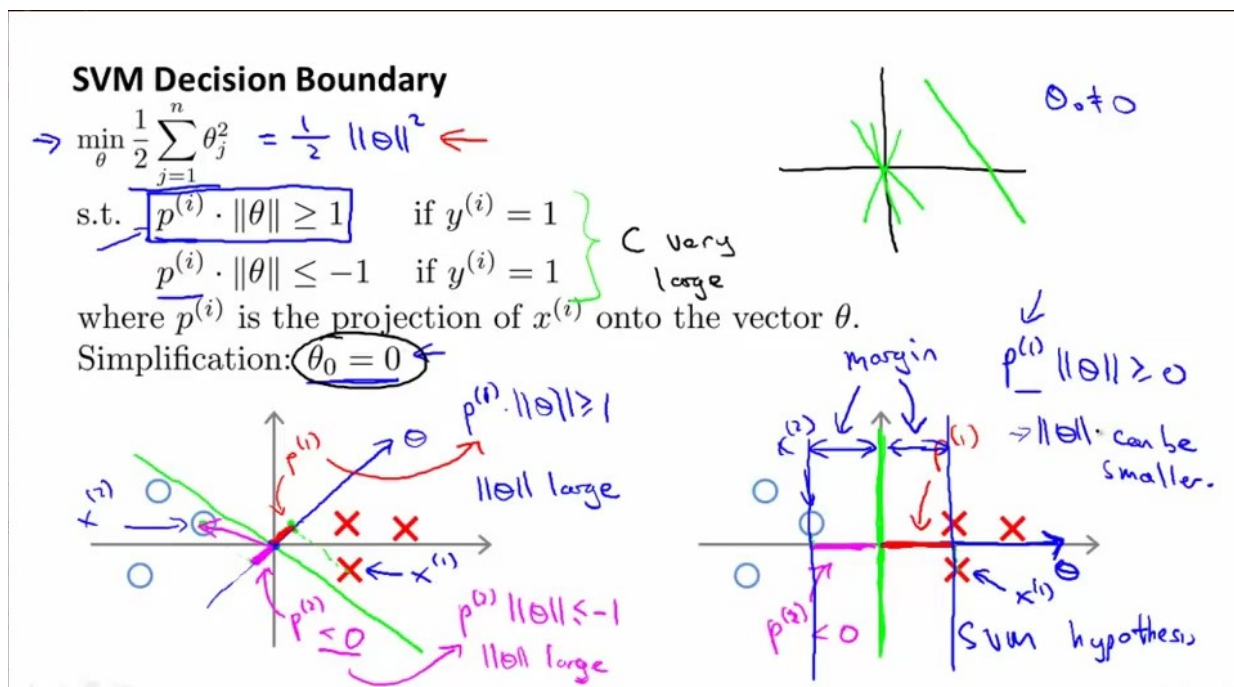
$$\theta^T x$$

$$= p^{(i)} \|\theta\|$$

이제 이 식을 좀 활용해 보자. C 가 매우 클때는 B 만 최소화 하면 되는데

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

이 식 자체가 *large margin* 을 찾아낸다. 왜 그런가 보면



(<http://blog.csdn.net/abcjennifer>)

왼쪽 그래프의 계산 과정을 보면 x_1 을 θ 에 *projection* 해서 얻은 p_1 이 매우 작다. 따라서 $p_1 * \|\theta\| \geq 1$ 에서 $\|\theta\|$ 가 커야 전체 식이 1보다 커지는데, 이러면 식 B 를 최소화 할 수 없다. 마찬가지로 p_2 는 매우 작은 음수고, $p_2 * \|\theta\| \leq -1$ 에서, $\|\theta\|$ 가 매우 큰 음수여야 한다. 이 또한 θ 를 크게 만드므로 식 B 가 작아지는 θ 를 찾지 못한다.

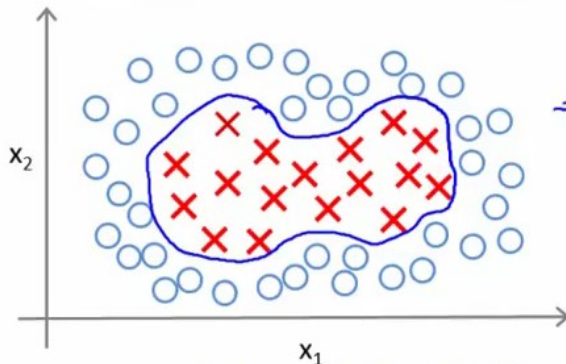
결국 p 가 커야만 θ 가 작아지기 때문에 p 를 크게 하는 θ 만 찾고, 이것은 *large margin* 이다. 따라서 초록색 같은 *low margin* 의 θ 는 선택되지 않는다.

정리하자면 C 가 매우 클때 SVM 은 *large margin* 을 찾고, 여기서 C 를 낮춤으로써 적당한 수준의 *classification* 을 얻을 수 있다.

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

Kernels

Non-linear Decision Boundary



Predict $y = 1$ if

$$\begin{aligned} \rightarrow & \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 \\ & + \theta_4 x_1^2 + \theta_5 x_2^2 + \dots \geq 0 \end{aligned}$$

$$h_{\theta}(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \dots \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$\rightarrow \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \dots$$

$$f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2, f_4 = x_1^2, f_5 = x_2^2, \dots$$

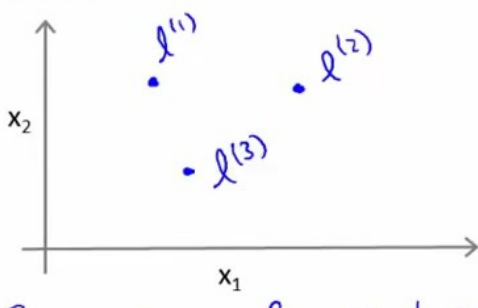
Is there a different / better choice of the features f_1, f_2, f_3, \dots ?

(<http://blog.csdn.net/abcjennifer>)

SVM 으로 *non-linear decision boundary* 를 어떻게 찾아낼까? 단순히 *high polynomial features* 를 사용하는 것보다 더 나은 방법은 없을까? 고차 다항식은 이미 지 처리 예제에서도 봤지만, 계산 비용이 너무 비싸다.

kernel 이란 개념이 있다.

Kernel



Given x , compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

$$\begin{aligned} & \|w\| \\ & \leftarrow \|x - l^{(1)}\|^2 \\ & \leftarrow \|x - l^{(2)}\|^2 \end{aligned}$$

Given x :

$$f_1 = \text{similarity}(x, x) = \exp\left(-\frac{2\sigma^2}{2\sigma^2}\right)$$

$$f_2 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$$f_3 = \text{similarity}(x, l^{(2)}) = \exp(\dots)$$

↑ ↑
kernel (Gaussian kernels) $k(x, l^{(1)})$

Andrew Ng

(<http://blog.csdn.net/abcjennifer>)

수동으로 몇몇 landmark l_1, l_2, \dots 을 고른후 이 landmark 사이의 거리로 새로운 feature f 를 만든다.

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

dl similarity function 을 kernel function 특히 여기서 사용한 수식은 gaussian kernel 이라 부른다.

Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^n (x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

$$f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

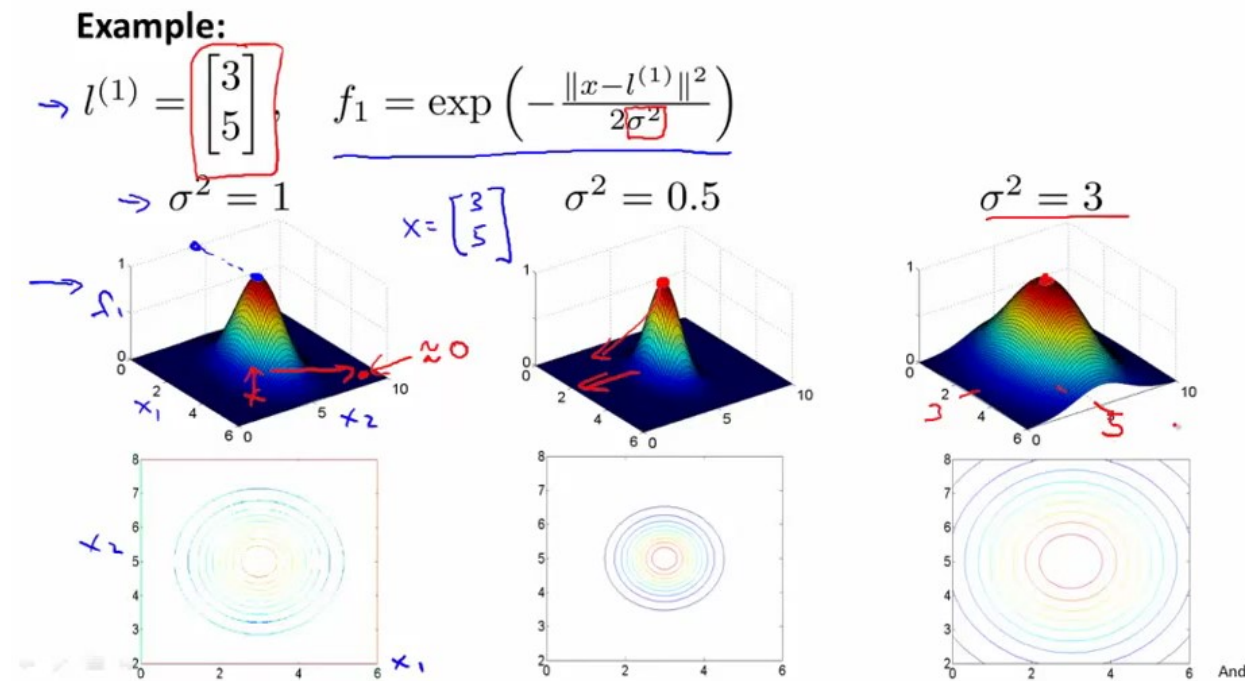
$$\begin{array}{lcl} l^{(1)} & \rightarrow & f_1 \\ l^{(2)} & \rightarrow & f_2 \\ l^{(3)} & \rightarrow & f_3 \end{array}$$

If x is far from $l^{(1)}$:

$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0.$$

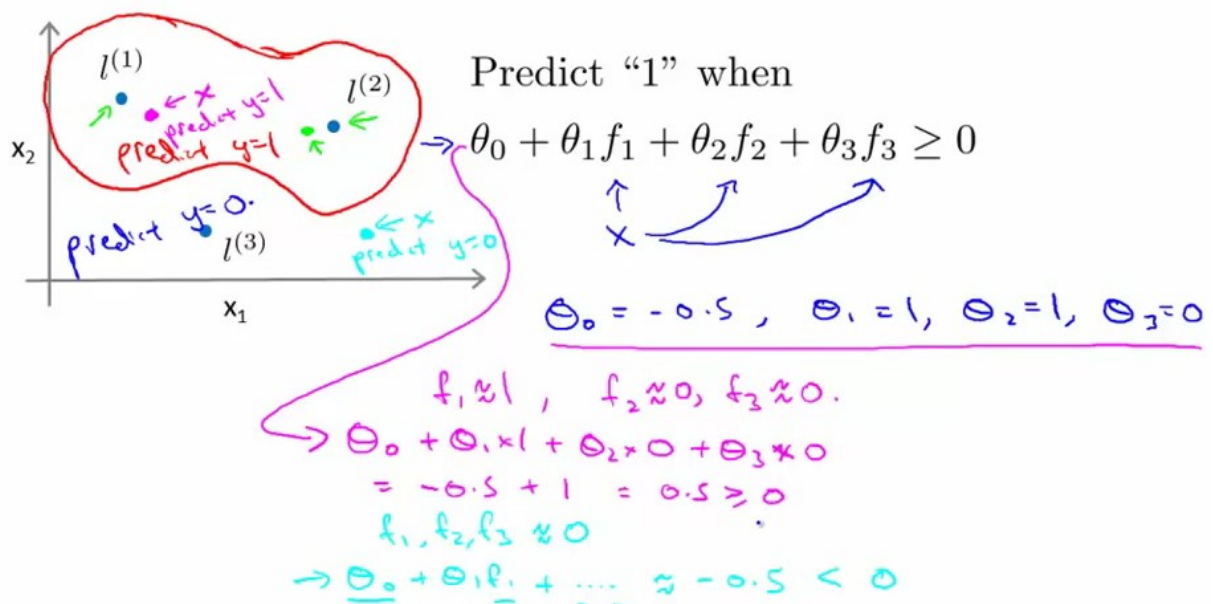
(<http://blog.csdn.net/abcjennifer>)

x 와 l 이 상당히 가까우면 f 는 1 에 근접하고, 상당히 멀면 0 에 가까워진다.



(<http://blog.csdn.net/abcjennifer>)

위 그림은 시그마에 따른 f 값의 변화를 보여주는데, 시그마가 작으면 작을수록 조금만 멀어도 f 값은 0 에 가까워진다.



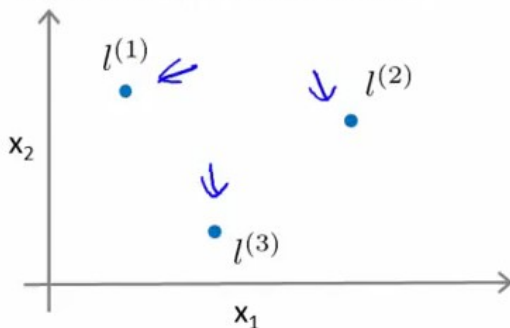
Andrew Ng

(<http://blog.csdn.net/abcjennifer>)

데이터가 *landmark* 중 하나에 라도 가까우면 적어도 하나의 f 가 1이 되어, $h(x)$ 가 1 이되고 반면 모든 *landmark* 에 멀면 모든 f 가 0 이 되어 $h(x)$ 가 0 이된다.

그럼 이제, 문제는 어떻게 *landmark* 를 정할 것인가?

Choosing the landmarks

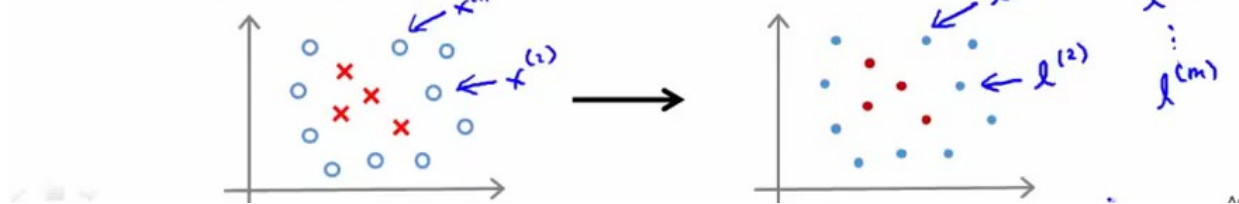


Given x :

$$\begin{aligned} \rightarrow f_i &= \text{similarity}(x, l^{(i)}) \\ &= \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right) \end{aligned}$$

Predict $y = 1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \dots$?



(<http://blog.csdn.net/abcjennifer>)

SVM with Kernels

- Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$,
- choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \dots, l^{(m)} = x^{(m)}$.

Given example x :

$$\begin{aligned} \rightarrow f_1 &= \text{similarity}(x, l^{(1)}) \\ \rightarrow f_2 &= \text{similarity}(x, l^{(2)}) \\ &\vdots \end{aligned}$$

$$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix} \quad f_0 = 1$$

For training example $(x^{(i)}, y^{(i)})$:

$$\begin{aligned} f_1^{(i)} &= \text{sim}(x^{(i)}, l^{(1)}) \\ f_2^{(i)} &= \text{sim}(x^{(i)}, l^{(2)}) \\ &\vdots \\ f_i^{(i)} &= \text{sim}(x^{(i)}, l^{(i)}) = \exp\left(-\frac{0}{2\sigma^2}\right) = 1 \end{aligned}$$

$$x^{(i)} \in \mathbb{R}^{n+1} \quad (\text{or } \mathbb{R}^n)$$

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$$

(<http://blog.csdn.net/abcjennifer>)

l_1, \dots, l_m 을 x_1, \dots, x_m 라 하자. 즉 각 *training example* 이 *landmark* 가 된다. 이를 이용해 구한 *feature vector* $f^{(i)}$ 중 하나는 $\text{sim}(x^{(i)}, l^{(i)})$ 이므로 1이 된다.

SVM with Kernels

Hypothesis: Given x , compute features $f \in \mathbb{R}^{m+1}$ $\theta \in \mathbb{R}^{m+1}$
 \rightarrow Predict "y=1" if $\theta^T f \geq 0$ $\theta_0 f_0 + \theta_1 f_1 + \dots + \theta_m f_m$

Training:

$$\rightarrow \min_{\theta} C \sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^m \theta_j^2$$

$\theta^T f^{(i)}$ $\theta^T f^{(i)}$ $\theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_m \end{bmatrix}$ (ignore θ_0) $m = 10,000$
 $\sum_j \theta_j^2 = \theta^T \theta \leftarrow \|\theta\|^2$
 $\theta^T M \theta$

Andrew

(<http://blog.csdn.net/abcjennifer>)

따라서 주어진 x 에 대해 $m+1$ 의 벡터 f 를 구해 $\theta^T f \geq 0$ 이면 $y = 1$ 이다.
 그리고 이 때 *feature* 수가 m 이 되므로

$$\min_{\theta} C \sum_{i=1}^m \text{cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^m \theta_j^2$$

마지막 항을 좀 자세히 보면

$$\sum_{j=1}^n \theta_j^2 \\ = \theta^T \theta$$

인데 SVM 실제 구현에서는 가운데 M 매트릭스를 삽입해 좀더 효율적으로 돌아가도록 한다. 이 M 은 어떤 *kernel* 을 사용하는지에 따라 다르다.

$$\theta^T M \theta$$

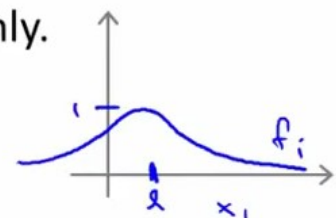
logistic regression 에 *kernel* 을 사용할 수도 있겠지만, 상당히 느리다. 반면 SVM 에서 는 마지막 항을 위 처럼 수정할 수 있기에 빠르게 동작한다.

Bias vs Variance in SVM

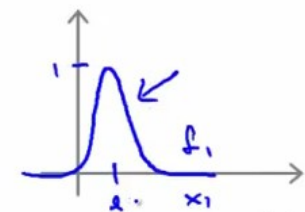
SVM parameters:

$C (= \frac{1}{\lambda})$. \rightarrow Large C: Lower bias, high variance. (small λ)
 \rightarrow Small C: Higher bias, low variance. (large λ)

σ^2 Large σ^2 : Features f_i vary more smoothly.
 \rightarrow Higher bias, lower variance.
 $\exp\left(-\frac{\|x - \mu^{(i)}\|^2}{2\sigma^2}\right)$



Small σ^2 : Features f_i vary less smoothly.
 Lower bias, higher variance.



Andrew

(<http://blog.csdn.net/abcjennifer>)

- (1) C 가 크면 *low bias, high variance* (== small λ)
- (2) C 가 작으면 *high bias, low variance* (== large λ)

σ 가 크면 f 가 적게 변하기 때문에 인풋 x 에 대해서도 *high bias, low variance* 다.

Using an SVM

Other choices of kernel

Note: Not all similarity functions $\text{similarity}(x, l)$ make valid kernels.

→ (Need to satisfy technical condition called “Mercer’s Theorem” to make sure SVM packages’ optimizations run correctly, and do not diverge).

Many off-the-shelf kernels available:

- Polynomial kernel:

$$k(x, l) = (x^T l + \text{constant})^{\text{degree}}$$

Handwritten examples: $(x^T l)^3$, $(x^T l + 1)^3$, $(x^T l + 5)^4$. Arrows indicate the degree of the polynomial.

- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ...

$$\text{sim}(x, l)$$

(<http://blog.csdn.net/abcjennifer>)

라이브러리를 사용하더라도 C 와 어떤 *kernel* 을 사용할지는 골라야 한다.

feature 가 크고, 트레이닝셋이 작을때는 *overfitting* 될 수 있으므로 *linear kernel* 을 사용하는 편이 낫다.

반면 n 이 작고, m 이 클 경우에는 *non-linear* 가설일 수 있으므로 *gaussian kernel* 을 사용할 수 있다. 그러면 σ 를 골라야 한다.

Kernel (similarity) functions:

function $f = \text{kernel}(\underline{x1}, \underline{x2})$

$$f = \exp\left(-\frac{\|\underline{x1} - \underline{x2}\|^2}{2\sigma^2}\right)$$

return

$$x \rightarrow \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{pmatrix}$$

→ Note: Do perform feature scaling before using the Gaussian kernel.

$$\rightarrow \|x - l\|^2$$

$$v = x - l$$

$$\|v\|^2 = v_1^2 + v_2^2 + \dots + v_n^2$$

$$= (x_1 - l_1)^2 + (x_2 - l_2)^2 + \dots + (x_n - l_n)^2$$

$x \in \mathbb{R}^n$
 1000 feet² 1-5 bedrooms

(<http://blog.csdn.net/abcjennifer>)

SVM 라이브러리를 이용할 때는 `kernel function` 을 직접 구현해야 한다. 이걸 이용해서 라이브러리는 `x` 에 대해 `f1, ..., fl` 을 계산한다.

만약에 *feature* 의 스케일이 다르면, `x1 = 10000, x2 = 5, ...` `$\|x - l\|^2$` 값이 숫자가 큰 항에 의해 좌우될 수 있으므로 *feature scaling* 을 하는편이 좋다.

Other choices of kernel

Other choices of kernel

Note: Not all similarity functions $\text{similarity}(x, l)$ make valid kernels.

→ (Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge).

Many off-the-shelf kernels available:

- Polynomial kernel:

$$k(x, l) = (x^T l + \text{const})^{\text{degree}}$$

$$(x^T l)^3, (x^T l + 1)^3, (x^T l + 5)^4$$

- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ...

$$\text{sim}(x, l)$$

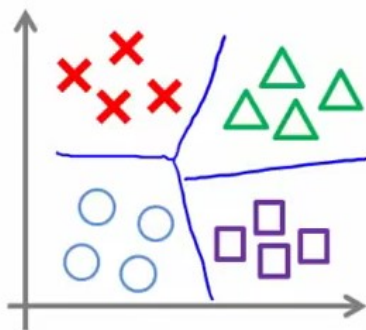
(<http://blog.csdn.net/abcjennifer>)

SVM 구현들이 계산을 최적화 하기위해 다양한 트릭을 이용한다. 이로 인해 모든 *similarity function* 유효한 커널이 되는건 아니고, “Mercer’s Theorem”을 만족해야만 한다. ~~인용하려 했는데 무슨말인지 모르겠음~~

그렇다고 커널이 *linear* 와 *gaussian* 만 있는건 아니고 다양한 커널이 있다. 그림을 참조하자.

Multi-class classification

Multi-class classification



$$y \in \{1, 2, 3, \dots, K\}$$

↑

Many SVM packages already have built-in multi-class classification functionality.

→ Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \dots, K$), get $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(K)}$
 Pick class i with largest $(\theta^{(i)})^T x$

↑ ↑ ... ↑
 $y=1$ $y=2$... $\theta=K$

(<http://blog.csdn.net/abcjennifer>)

대부분의 SVM 라이브러리들은 *multi-class* 에 대한 함수를 제공한다. 그러나 이것들을 사용하는 대신 *one-vs-all* 방법을 사용할 수도 있다. k 개의 클래스가 있다면 k 개의 SVM 훈련시키면 된다.

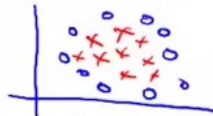
Logistic regression vs SVM

Logistic regression vs. SVMs

n = number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples

- If n is large (relative to m): (e.g. $n \geq m$, $n = 10,000$, $m = 10 \dots 1000$)
- Use logistic regression, or SVM without a kernel (“linear kernel”)

- If n is small, m is intermediate: ($n = 1-1000$, $m = 10-10,000$) ←
- Use SVM with Gaussian kernel
- If n is small, m is large: ($n = 1-1000$, $m = 50,000+$)
- Create/add more features, then use logistic regression or SVM without a kernel ↑
- Neural network likely to work well for most of these settings, but may be slower to train.



(<http://blog.csdn.net/abcjennifer>)

- (1) $n \geq m$ 이면 *logistic regression* 이나 *linear kernel* 이 낫다.
- (2) n 이 작고, m 이 중간 사이즈면 *gaussian kernel* 을
- (3) n 이 작고 m 이 크면 *gaussian* 은 상당히 느려진다. *feature* 를 좀 수정하고, *logistic* 이나 *linear kernel* 을 이용한다.

SVM 의 장점은 다양한 *kernel* 을 *non-linear function* 을 훈련시키기 위해 사용할 수 있다는 점이다.

References

- (1) *Machine Learning* by **Andrew NG**
- (2) <http://blog.csdn.net/linuxcumt>
- (3) <http://blog.csdn.net/abcjennifer>

0 Comments lambda

 Login ▾ Recommend  Share

Sort by Best ▾



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Be the first to comment.

ALSO ON LAMBDA

10분만에 Github Profile 만들기

6 comments • 2 years ago

sMiLo — Windows 환경에서 nvm-windows 설치 후 진행하니 아래 에러가 발생 했습니다.\$ omg init githubsmilo oh-**하스켈로 배우는 함수형 언어 4**



1 comment • a year ago

Junmo Roger Kang — 아이고 공부가 드디어 모나드에서 걸리네요 ㅜㅜ**CLOS, Common Lisp Object System**

1 comment • 4 years ago

Philippe Dallaire — i'm drunk and was searching for something else about lisp but this answered questions I never**HOME**

1 comment • a year ago

Eunju Amy Sohn — 공부에 많은 도움을 받고 있습니다. 양질의 포스팅을 많이 올려주셔서 감사합니다! Subscribe  Add Disqus to your siteAdd DisqusAdd  Privacy